Proceedings of the Second International Workshop on Sustainable Ultrascale Computing Systems (NESUS 2015)
Krakow, Poland

Jesus Carretero, Javier Garcia Blas
Roman Wyrzykowski, Emmanuel Jeannot.
(Editors)

September 10-11, 2015

Log File Analysis in Cloud with Apache Hadoop and Apache Spark

Ilias Mavridis
Aristotle University of Thessaloniki, Greece
imavridis@csd.auth.gr

Eleni Karatza
Aristotle University of Thessaloniki, Greece
karatza@csd.auth.gr

Abstract

Log files are a very important set of data that can lead to useful information through proper analysis. Due to the high production rate and the number of devices and software that generate logs, the use of cloud services for log analysis is almost necessary. This paper reviews the cloud computational framework Apache™ Hadoop®, highlights the differences and similarities between Hadoop MapReduce and Apache Spark™ and evaluates the performance of them. Log file analysis applications were developed in both frameworks and performed SQL-type queries in real Apache Web Server log files. Various measurements were taken for each application and query with different parameters in order to extract safe conclusions about the performance of the two frameworks.

Keywords Log analysis, Cloud, Apache Hadoop, Apache Spark, Performance evaluation

I. Introduction

The log files are a rich source of information that can be used for various purposes. However, the high production rate and the diversity between the logs makes it difficult to analyze. The log production rate can reach to several TeraBytes (TB) or PetaBytes (PB) per day, for example Facebook dealt with 130 TB of logs every day [1] in 2010 and in 2014 they have stored 300 PB of logs [2]. For these reasons conventional database solutions can't be used for the analysis, but cloud or even interconnected cloud systems [3] required to achieve scalability and elasticity. Many big companies like Facebook, Amazon, ebay, etc. use cloud computing to analyze logs. Also from academia there are many papers which investigate cloud computing (mainly Hadoop) to analyze logs [4] - [14].

Hadoop is the framework that has mainly been used to store and analyze data. Hadoop was designed for batch processing providing scalability and fault tolerance but not fast performance [15]. It enables applications to run in thousands of nodes with Petabytes of data. Hadoop responds to the large amount of logs by breaking up log files into blocks and distribute them to the nodes of the Hadoop cluster. It follows a similar strategy for computing by breaking jobs into a number of smaller tasks that will be executed in nodes of the cluster.

However, Hadoop's performance is not suitable for real-time applications [16] because it frequently writes and reads data from the disk. Spark solves this problem by minimizing these data transfers from and to disk by using effectively the main memory and performing in-memory computations. Also it provides a new set of high-level tools for SQL queries, stream processing, machine learning and graph processing [17].

Our work complements existing research by investigating and comparing log file analysis in Hadoop and Spark. The rest of the paper is organized as follows. Section II provides an overview of related research. Section III describes briefly what is log file and log file analysis in cloud. Section IV outlines the two open source computing frameworks Hadoop and Spark. Section V describes the setting of our experiments. Section VI presents the experimental results. Finally Section

VII concludes this paper.

II. Related Work

There are many papers whose authors investigate and propose the use of cloud computing to log file analysis. Paper [4] discusses the differences between the traditional relational database and big data. The authors claim that log files were produced in higher rate than traditional systems can serve and show experimental log file analysis through Hadoop cluster.

Paper [5] presents a weblog analysis system based on the Hadoop HDFS, Hadoop MapReduce and Pig Latin Language. The system aims to assist administrator to quickly analyze data and take business decisions. It provides an administrators monitoring system, problem identification and system's future trend prediction.

Also in [6] the authors discuss a Hadoop based system with Pig for web log applications. A web application has been created to distributed store log files on the Hadoop cluster, run MapReduce jobs and display results in graphical formats like bar charts. They have conclude that by this way there is a significant response time improvement and that MapReduce can successfully and efficiently process large datasets.

In line with [5] and [6], paper [7] proposes a mass log data processing and data mining method based on Hadoop to achieve scalability and high performance. To achieve scalability and reliability the log data are stored in HDFS and it is used Hadoop's MapReduce for high performance. In this case also, the experimental results show that the Hadoop based processing improves the performance of querying.

The paper [8] presents a scalable platform named Analysis Farm, for network log analysis, fast aggregation and agile query. To achieve storage scale-out, computation scale-out and agile query, OpenStack has been used for resource provisioning, and MongoDB for log storage and analysis. In experiments with Analysis Farm prototype with 10 MongoDB servers, the system managed to aggregate about 3 million log records in a 10-minute interval time and effectively query more than 400 million records per day.

A Hadoop based flow logs analyzing system has been proposed in paper [9]. This system uses for log analysis a new script language called Log-QL, which is a SQL-like language. After experiments they concluded that their distributed system is faster than the centralized system.

Paper [10] presents a cloud platform for batch log data analysis with Hadoop and Spark. The authors propose a cloud platform with batch processing and in-memory computing capabilities by combining Hadoop, Spark and Hive/Shark. The proposed system manage to analyze logs with higher stability, availability and efficiency than standalone Hadoop-based log analysis tools.

In paper [11] has been implemented a Hadoop MapReduce-based framework to analyze logs for anomaly detection. First they collect the system logs from each node of the monitored cluster to the analysis cluster. Then, they apply the K-means clustering algorithm to integrate the collected logs. After that, they execute a MapReduce-Based algorithm to parse these clustered log files. By this way, they can monitor the distributed cluster status and detect its anomalies.

Log file analysis can also be used for system threats and problem identification. Paper [12] presents a new approach which uses a MapReduce algorithm for log analysis to provide appropriate security alerts or warnings. They achieve a significant improvement in response time for large log file analysis and as a result to a faster reaction by the administrator.

In [13] the authors describe an approach which uses log file analysis for intrusion detection. The objective of the paper is to enhance the throughput and scalability by using Hadoop MapReduce and cloud computing infrastructure. They describe the architecture and implement performance analysis of an intrusion detection system based on Cloud Computing. From the experiments they conclude that the system fulfills the scalability, fault tolerant and reliability expectations which is designed for.

Finally [14] presents SAFAL, a Spatio-temporal Analyzer of FTP Access Logs collected by UNAVCO's data center. These logs contain massive amounts of data like borehole seismic, strainmeter, meteorological, and digital imagery data. The system was developed using MapReduce/Hadoop in order to identify trends in GPS data usage. They conducted several experiments and found that SAFAL was able to analyze very efficiently millions of lines of FTP access logs. Also

the authors conclude that it could be possible to create near real time maps by the analysis of the logs.

III. Log File Analysis

Log file analysis is the analysis of log data in order to extract some useful information [17]. As the log data come from many different systems in a variety of forms, a proper analysis requires a good knowledge of the system. It must be clear what is good and bad for a specific system and what is suspicious or not. Worth noting that a same value maybe is suspicious for a system but completely normal for another one.

III.1 Log Files

Each working computer system collects information about various operations. This information is stored in specific files which called log files [19]. Log files consist of log messages or simply log(s). A log message is what a computer system, device, software, etc. generates in response to some sort of stimuli [18]. The information that pulled out of a log message and declares why the log message generated is called log data [18].

A common log message includes the timestamp, the source, and the data. The timestamp indicates the time at which the log message was created. The source is the system that created the log message and the data is the essence of the log message. Unfortunately this format is not a standard and so the log message can be significantly different from system to system.

```
1 192.168.100.252 - - [31/Jul/2014:09:30:23 +0700] "GET /p4p/report_detail_edit.php?name_t=3 HTTP/1.1" 200 8013
2 110.168.229.112 - - [31/Jul/2014:09:30:25 +0700] "GET /p4p/detail.php HTTP/1.1" 200 5433
3 110.168.229.112 - - [31/Jul/2014:09:30:25 +0700] "GET /p4p/jquery/jquery.calendars.persian.js HTTP/1.1" 404 324
```

Figure 1: Apache web access log.

One of the most widespread types of log files in the web is the log files that were produced by the Apache HTTP Server [20]. Figure 1 shows the first three lines of a real Apache web access log file.

As shown in Figure 1, the first element of every row is the ip address of the client (e.g. 192.168.100.252) or may be the name of the node which made the request to the server. Then there are two dashes (- -) which means that there is no value for this two fields [20]. The first dash stands for the identity of the client specified in RFC 1413 [21], and the second dash represents the user id of the person requesting the server. The fourth element in each of these log messages indicates the date and time that the client's request had been served by the server and in the same brackets there is the server's time zone (e.g. +0700). Next in double quotes is the request line from the client. The request line first contains the method that has been used by the client (e.g. GET), then there is the requested source (e.g./`p4p/report_detail_edit.php?name_t=3`) and finally the used protocol (e.g. HTTP/1.1). At the end of each row there are two numbers that follows the request line.The first number is the status code that the server returns to the client (e.g. 200) and the last number indicates the size of the object returned to the client and is usually expressed in bytes (e.g. 8013).

III.2 Log Analysis in the Cloud

The rise of cloud computing and the growing requirement for processing and storing capabilities for log analysis, resulted in the combination of cloud computing and log analysis. It has emerged a new term, the Logging as a Service (LaaS). There are some cloud service providers that undertake to analyze the logs for one of their clients [22]. Users of such services can collect logs from various devices and software and submit them to the cloud for processing, as shown in Figure 2. The LaaS is a quite new cloud service but there are already providers like [23] and [24] that offer different service products. There are some features and capabilities common to all and some others that vary from provider to provider.

The main elements that LaaS has is the file uploading from the user to the cloud, indexing of data (for fast search, etc.), long-term storage and a user interface to search and review the data [18]. Most providers also supports various types of log formats and have their own API [23] [24]. Moreover, the majority of providers charge their services with the model "pay as you go", where the user is charged depending on the use of services has made.

On the other hand, there are differences in the way that each provider has developed its system [18]. Some providers have built their services to another

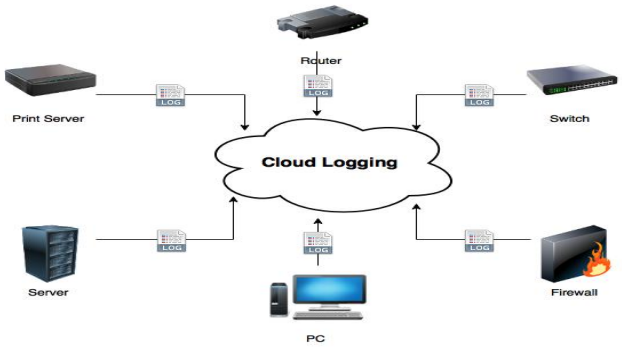

Figure 2: Logging as a Service (LaaS).

provider's cloud, while others in their own cloud infrastructure. Also, although all LaaS providers offer the possibility of long-term storage of data, the charges are not the same and the highest possible storage time differs also. Furthermore as it's expected the charges vary from provider to provider.

IV. The Computational Frameworks

The cloud computing frameworks that have been reviewed are the Hadoop and Spark. Hadoop is a well-established framework that has been used for storing, managing and processing large data volumes for many years by companies like Facebook, Yahoo, Adobe, Twitter, ebay, IBM, Linkedin and Spotify [25]. Hadoop is based on MapReduce programming model which is developed for batch processing. However the need for real-time data analysis leads to a new general engine for large-scale data processing, Spark. Spark was developed by AMPLab [26] of UC Berkeley and unlike the Hadoop's MapReduce, it uses the main memory, achieving up to 100 times higher performance for certain applications compared to Hadoop MapReduce [27].

IV.1 Apache Hadoop

Hadoop [28] is a framework for running applications on large clusters built of commodity hardware. It comes from Apache Nutch [29], which is an open source search engine. Key element to develop Hadoop were two Google papers, the first one was published in 2003 and describes the Google Distributed Filesystem -GFS [30] and the second one was published in 2004, and describes the MapReduce [31]. In February 2006 a part of Nutch became independent and created Hadoop. In 2010 a team from Yahoo began to design the next generation of Hadoop, the Hadoop YARN (Yet Another Resource Negotiator) or MapReduce2 [32].

The YARN changed the resource management to overcome the problems that had arisen, and also made the Hadoop capable of supporting a wide range of new applications with new features. The YARN is more general than the MapReduce (Figure 3), in fact the MapReduce is a YARN application. There are other YARN applications like Spark [33], which can run parallel to the MapReduce, under the same resource manager.

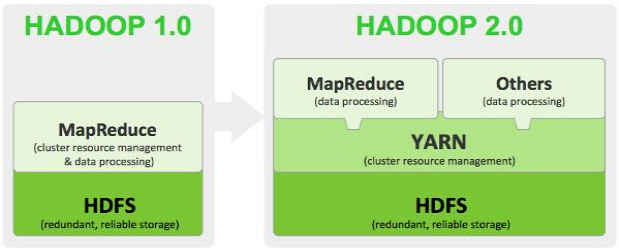

Figure 3: Hadoop 1.0 to Hadoop 2.0 [33].

Hadoop at its core lies at the HDFS [34] and the MapReduce computational model. However the term is also used for a set of related programs that used for distributed processing and processing of large-scale data, such as *Hive*^{TM} [35], *Mahout*^{TM} [36], *Zookeeper*^{TM} [37] and others.

IV.1.1 Hadoop Distributed File System

The Hadoop distributed file system (HDFS) is created as a file system with blocks. As shown in Figure 4, the files are separated into blocks of a fixed size and stored at different nodes of Hadoop cluster [34]. Because the files are divided into smaller blocks, HDFS can store files much bigger than the disk capacity of each node. The stored files follows the write-once, read-many approach and can not be modified. On the other

hand there are the metadata files that describe the system and are changeable. There is a dedicated node called NameNode that stores all system's metadata and ensures that is always up to date.

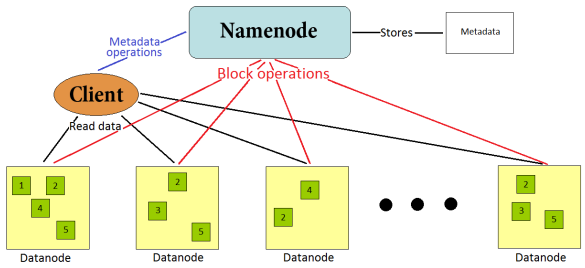

Figure 4: HDFS architecture.

The HDFS is implemented with the master/slave model. The NameNode is the master that manages the file system namespace and determines the client's access to the files. The slaves are called DataNodes and are responsible for storing the data and do anything that NameNode dictates them.

For fault-tolerance, HDFS replicates each block of a DataNode to other nodes [38]. To prevent disaster from Namenode failure, there is a secondary NameNode and replicas of the NameNode's metadata. Also worth noting that HDFS tries to respond to a read request with the closer copy to the reader (Rack Awareness) [34] in order to minimize the total bandwidth utilization and the reading time.

IV.1.2 MapReduce

MapReduce is a batch-based, distributed computing framework presented by Google's paper [31]. A MapReduce program consist of the Map Phase and the Reduce Phase [39]. Initially the data are processed by the map function and produce an intermediate result in the form of <Key, Value>. There can be many values with the same key. After that follows the reduce function. The reduce function performs a summary operation that process the intermediate results and generates the final result.

In the original version of the Hadoop MapReduce there are two types of nodes, the JobTracker (master) and TaskTrackers (slaves). In each MapReduce cluster there is a JobTracker that is responsible for resource management, job scheduling and monitoring [40]. The TaskTrackers run processes that were assigned to them by the JobTracker.

With Hadoop Yarn the execution model became more scalable and generic than the earlier version. The new Hadoop Yarn can run applications that do not follow the MapReduce model. With YARN, there is no longer a single JobTracker that does all the resource management, instead the ResourceManager and the NodeManager manage the applications. The ResourceManager is allocating resources to the different applications of the cluster. The ApplicationMaster negotiates resources from the ResourceManager and works with the NodeManager(s) to execute and monitor the component tasks [32].

IV.2 Apache Spark

Spark was developed in 2009 by AMPLab of UC Berkeley, and became an open source project in 2010 [41]. In 2013, the program was donated to the Apache software foundation and in February 2014 the Spark was a high-level program in the same foundation [42]. In November 2014, the engineering team at Databricks set a new record in large-scale sorting using Spark [43].

Spark extends the popular MapReduce model, supports more types of data processing and the combination of them, such as SQL-type queries and data flow processing. For ease of use, Spark has Python, Java, Scala and SQL APIs, and many embedded libraries.

One of the main features of Spark is the exploitation of main memory [44]. It may accelerate an application to one hundred times using memory and ten times using only the disc compared to Hadoop MapReduce cluster [41].

IV.2.1 Spark Ecosystem

Spark is a general purpose engine that supports higher-level items specialized to a particular kind of processing [41]. These components are designed to operate close to the core, and can be used as libraries during the development of a program.

The components of the Spark ecosystem are [41]:

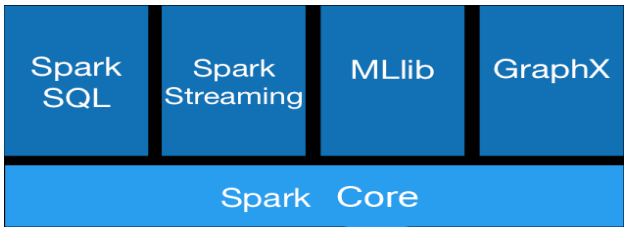

Figure 5: Spark ecosystem.

- Spark Core: Is the general execution engine for the Spark platform and every other functionality is built on top of it.
- Spark SQL: Is a Spark module for structured data processing. It can act as a distributed SQL query engine.
- Spark Streaming: Enables interactive and analytical applications across both streaming and historical data.
- MLlib: Is a scalable machine learning library.
- GraphX: Is a graph computation engine that enables users to manipulate and perform parallel processing in graphs.

IV.2.2 Resilient Distributed Dataset

Spark uses a new parallel and fault-tolerant data structure called Resilient Distributed Dataset (RDD) [45]. Spark automatically distributes the RDD data in the cluster and performs parallel processing on them. The RDDs can contain any object or class of Python, Java or Scala.

The RDDs supports two types of operations, transformations which generate a new dataset from an existing one, and actions which return a value after running a computation on a dataset [46]. For example, map is a transformation that passes each element of a RDD to a function and results to a new RDD with the computed values. On the contrary, reduce is an action that passes each element of a RDD to a function and returns a single value as a result.

To achieve efficiency Spark's transformations are "lazy" [47], which means that the computation of a new RDD is not executed immediately after the command is given. Instead, the transformations run only when an action needs the transformation result. On the other hand actions run immediately.

One of the most important capabilities of Spark is persisting or caching a dataset in main memory [47]. By maintaining a RDD in main memory, each node can perform much faster future computations in this dataset, often more than ten times faster. Also the cached RDD is fault-tolerant, which means that if a partition of RDD is damaged, then it will automatically recalculated with the proper transformations and will be replaced.

V. Experimental Setup

We have conducted a series of tests to experimentally evaluate the performance of the two frameworks. For this purpose has been developed a cluster with virtualized computing resources of Okeanos [48]. Okeanos is an IaaS (Infrastructure as a Service) Service, developed by the Greek Research and Technology Network [49]. It is offered to the Greek Research and Academic community and provides access to Virtual Machines, Virtual Ethernets, Virtual Disks, and Virtual Firewalls, over a web-based UI.

In these experiments have been used 5 nodes, 4 slaves and 1 master. The slaves have configured with 2 cpu units, 6GB memory, 40GB disk space and the master with 8 cpu units, 8GB memory and 40GB disk space.

The log file that has been used to testing is a real world log file. This file is an Apache HTTP Server log which is accessible through internet and was found after a relevant search. The log messages of this file have the form shown in Figure 1. To perform the experiments the two frameworks were installed to the cluster in the same nodes. The programs were developed in the same language (java) for both frameworks and the log files were saved in HDFS.

VI. Experimental Results

For each program, measurements were taken related to the execution time and the number of active slave nodes, the size of the input file and the type of program.

VI.1 Hadoop Expirements

We conducted different tests with different input file sizes, number of active nodes and programs. As shown in Figure 6 and Figure 7 the increment of the input file size results in the increment of the program's execution time. We also observe an increment in execution time when the number of active nodes is reduced, with a particularly big increment when remains only one slave node. These observations is completely reasonable because by these ways the processing volume for each node has been increased and as a result the processing time.

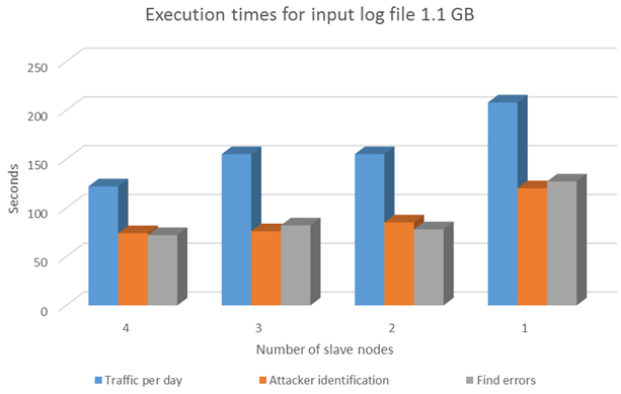

Figure 6: Execution times of Hadoop programs with 1.1GB input file.

Furthermore we see that the first program (blue) takes considerably more time to run. This is due to the nature of the program, because its Reduce phase requires much more processing work than the Reduce phase of the other two programs. And while Map processes have almost the same execution times, the big difference in Reduce process makes a difference in the end.

Moreover we observe that there is a slight difference in execution times between two or four nodes for the smaller file. This makes sense because the file is relatively small and two nodes have enough computing power to execute the required processes. On the other hand for the same reason we see that for the largest file each additional node makes a difference in execution time.

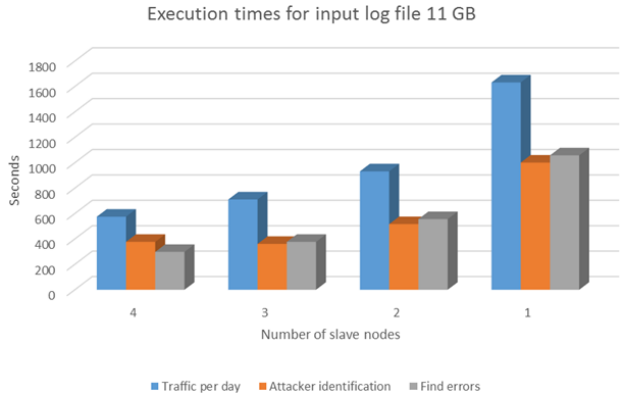

Figure 7: Execution times of Hadoop programs with 11GB input file.

Finally in Figure 7 all four nodes were better exploited due to the large input file. At this case the doubling of the active nodes leads to almost halve of the execution time (differs for program to program).

VI.2 Spark Expirements

In correspondence to Hadoop's experiments, relevant experiments carried out in Spark. Spark programs can run as a standalone Spark applications or executed on YARN. For the following tests the programs run as standalone Spark applications (both are supported from the developed system).

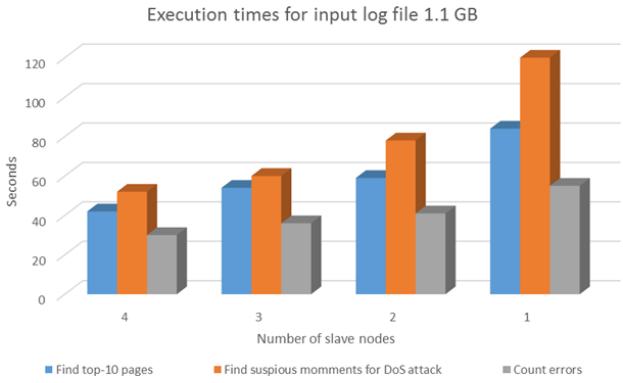

Figure 8: Execution times of Spark programs with 1.1GB input file.

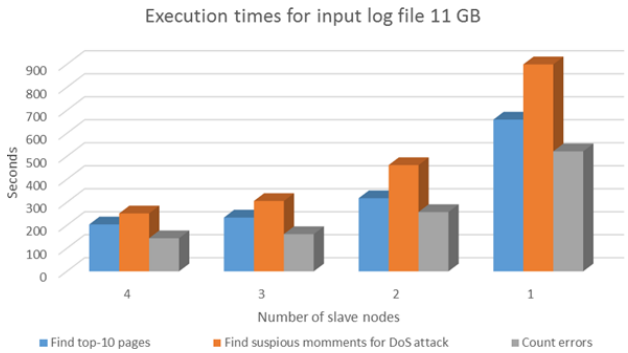

Figure 9: Execution times of Spark programs with 11GB input file.

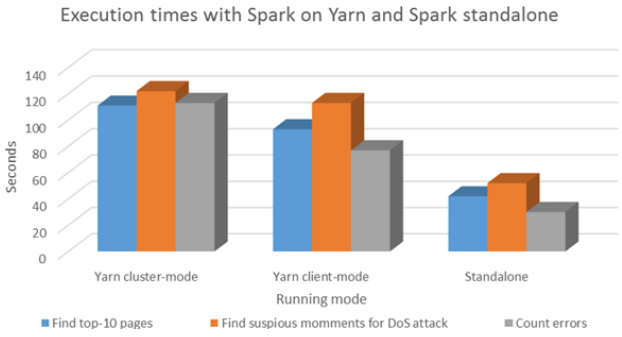

Figure 10: Spark on yarn and Spark standalone.

In these tests we generally observed that Spark's behavior is same as Hadoop. The increment of the size of the input file or the reduce of active slaves increase the program execution time especially when reamains active only one node. This is reasonable because -as explained previously- with these two ways the processing volume for each node has been increased and hence the processing time.

Furthermore in Figure 8 and Figure 9 we see that the three programs that were tested have different execution times, and even maintain the finish order. The third program (gray) has the less execution time, follows the first (blue) and finally the second (orange). This is because the programs require a different number and type of calculations, so the simpler program finishes first.

Also we observe that for input file of 1.1 GB there is a small difference in execution time with two to four nodes, because the file is relatively small and the process required can be carried out with two nodes. Same as Hadoop, for large file of 11 GB, each additional node makes a difference by contributing to the execution process.

In addition these programs are executed with the same input file in the same cluster but in a different way. Figure 10 shows the difference in the execution time according to whether the programs run on YARN or standalone. The execution of Spark programs on YARN offers additional features such as monitoring, dynamic resource management of the cluster, security through Kerberos protocol, possibility of parallel execution of various programs (e.g. MapReduce, Hive) and other features that are not supported by the standalone mode. However, as shown in Figure 10 the execution of programs on YARN is quite slower than standalone, that is because YARN has a quite complex resource management and scheduling compared to the Spark standalone and as a result there is a difference in execution time.

As shown in Figure 10, there are two types of yarn modes. In cluster-mode the driver runs in a process of the master who manages YARN. On the contrary in client-mode driver runs on client's process [50].

VI.3 SQL-type Queries Experiments

Figure 11 presents the performance comparison of Hadoop Hive and Spark SQL which are used for sql-type queries. For the experiments that we have contacted the Spark SQL runs in standalone mode and the executed queries are about error counting. The execution time of Spark SQL improved significantly when the table saved in main memory with the command CACHE TABLE tableName. As shown in Figure 11 the performance of Spark SQL is better than Hive. This happens because Spark SQL has a set of techniques to prevent reads and writes to disk storage, caching of tables in memory and optimizing efficiency.

VI.4 Overall Results

Various measurements have shown how systems react to changes in the size of the input file, the type of

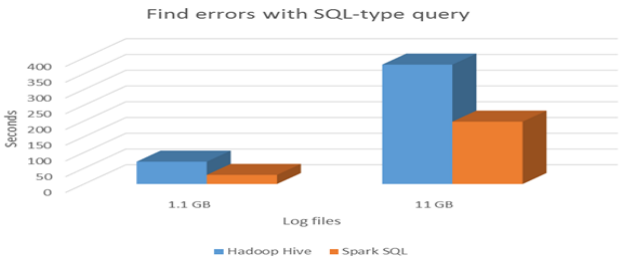

Figure 11: Execution times of SQL-type queries.

executed program and available nodes in the cluster. The two frameworks are highly configurable and their performance may vary depending on their settings. However by mainly maintaining presets and executing the same program with the same input data we can draw a conclusion.

To directly compare the two frameworks were executed on both the same programs. The first program is about error counting and the second is about errors finding. Figure 12 shows the performance of the frameworks for the first program, for input files of 1.1 GB and of 11 GB. The Spark presents the best performance, follows the Spark SQL, and then Hadoop MapReduce and Hadoop Hive with big difference in execution times. These results are in complete agreement with what has been previously described and confirm that the performance of Spark in most cases is better than Hadoop. The same conclusion comes from Figure 13 showing the performance of the second program for both frameworks.

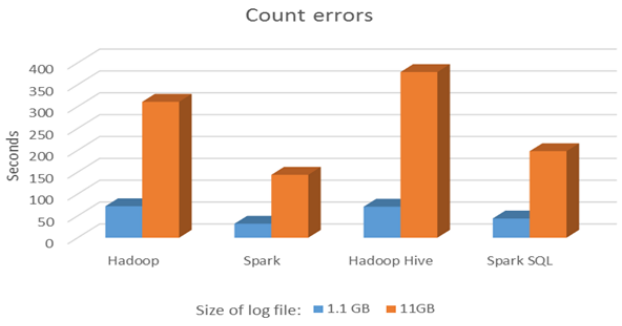

Figure 12: Execution times of count errors programs and SQL-type queries.

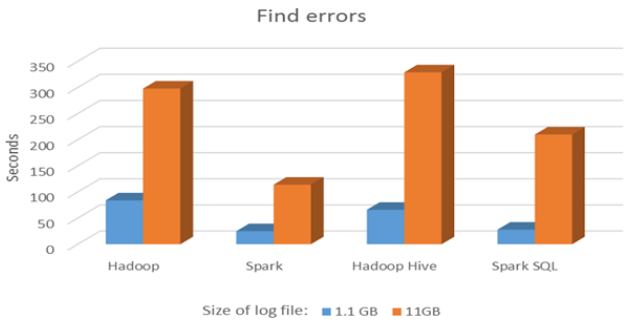

Figure 13: Execution times of find errors programs and SQL-type queries.

VII. Conclusions

This work aims to investigate the analysis of log files with the two most widespread computing frameworks in cloud computing, the well-established Hadoop and rising Spark. In the two frameworks have developed, executed and evaluated realistic programs for analyzing logs.

The two frameworks have the common goal of parallel processes execution on distributed files or other input files. The Hadoop is one of the first frameworks for cloud computing, is widely used and is one of the most active projects of the Apache foundation. Over the years Hadoop evolved and improved in order to meet the new era needs. These new needs led also to the creation of Spark.

Spark is different from Hadoop's MapReduce to two key points, which give Spark better performance and flexibility. The first is that Spark saves intermediate results in memory instead of the disk, thus it dramatically reduces the execution time. Secondly, Spark except of MapReduce functions, supports a wide range of new capabilities that can be combined to generate new powerful programs.

The various experiments that have been carried out show Spark's best performance. However, the programs were implemented in such a way to make possible the comparison between the two frameworks. As future work could be implemented programs that make full use of Spark capabilities in order to evaluate the performance of the framework for more complex

log analysis programs.

Acknowledgment

The authors would like to thank Okeanos the GRNET's cloud service for the valuable resources.

References

[1] https://www.facebook.com/notes/facebook-engineering/scaling-facebook-to-500-million-users-and-beyond/409881258919

[2] https://code.facebook.com/posts/229861827208629/scaling-the-facebook-data-warehouse-to-300-pb/

[3] I.A. Moschakis and H.D. Karatza, "A meta-heuristic optimization approach to the scheduling of Bag-of-Tasks applications on heterogeneous Clouds with multi-level arrivals and critical jobs," *Simulation Modelling Practice and Theory*, Elsevier, vol. 57, pp. 1-25, 2015.

[4] B. Kotiyal, A. Kumar, B. Pant and R. Goudar, "Big Data: Mining of Log File through Hadoop," in *IEEE International Conference on Human Computer Interactions (ICHCI'13)*, Chennai, India, August 2013, pp. 1-7.

[5] C. Wang, C. Tsai, C. Fan, Sh. Yuan, "A Hadoop based Weblog Analysis System," in *7th International Conference on Ubi-Media Computing and Workshops (U-MEDIA 2014)*, Ulaanbaatar, Mongolia, July 2014, pp. 72-77.

[6] S. Narkhede and T. Baraskar, "HMR log analyzer: Analyze web application logs over Hadoop MapReduce," *International Journal of UbiComp (IJU)*, vol.4, No.3, pp. 41-51, 2013.

[7] H. Yu and D.i Wang, "Mass Log Data Processing and Mining Based on Hadoop and Cloud Computing," in *7th International Conference on Computer Science and Education (ICCSE 2012)*,Melbourne, Australia, July 2012, pp. 197.

[8] J. Wei, Y. Zhao, K. Jiang, R. Xie and Y. Jin, "Analysis farm: A cloud-based scalable aggregation and query platform for network log analysis," in *International Conference on Cloud and Service Computing (CSC)*, Hong Kong, China, December 2011, pp. 354-359.

[9] J. Yang, Y. Zhang, S. Zhang and Dazhong He, "Mass flow logs analysis system based on Hadoop," in *5th IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT)*, Guilin, China, November 2013, pp. 115-118.

[10] X. LIN, P. WANG and B. WU, "Log analysis in cloud computing environment with Hadoop and Spark," in *5th IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT 2013)*, Guilin, China, November 2013, pp. 273-276.

[11] Y. Liu, W. Pan, N. Cao and G. Qiao, "System Anomaly Detection in Distributed Systems through MapReduce-Based Log Analysis," in *3rd International Conference on Advanced Computer Theory and Engineering (ICACTE)*, Chengdu, China, August 2010, pp. V6-410 - V6-413 .

[12] S. Vernekar and A. Buchade, "MapReduce based Log File Analysis for System Threats and Problem Identification," in *Advance Computing Conference (IACC), 2013 IEEE 3rd Internationa*l, Patiala, India, February 2013, pp. 831-835.

[13] M. Kumar and Dr. M. Hanumanthappa, "Scalable Intrusion Detection Systems Log Analysis using Cloud Computing Infrastructure," in *2013 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, Tamilnadu, India, December 2013, pp.1-4.

[14] H. Kathleen and R. Abdelmounaam, "SAFAL: A MapReduce Spatio-temporal Analyzer for UNAVCO FTP Logs," in *IEEE 16th International Conference on Computational Science and Engineering (CSE)*, Sydney, Australia, December 2013, pp. 1083-1090.

[15] http://wiki.apache.org/hadoop/

[16] G.L. Stavrinides, H.D. Karatza, "A cost-effective and QoS-aware approach to scheduling real-time workflow applications in PaaS and SaaS clouds," in *3rd International Conference on Future Internet of*

Things and Cloud (FiCloud'15), Rome, Italy, August 2015, pp. 231-239.

[17] `https://databricks.com/spark`

[18] Dr. A. A. Chuvakin, K. J. Schmidt, Chr. Phillips, P. Moulder, *Logging and Log Management: the authoritative guide to understanding the concepts surrounding logging and log management*, Elsevier Inc. Waltham, 2013.

[19] J. Pinto Leite, "Analysis of Log Files as a Security Aid," in *6th Iberian Conference on Information Systems and Technologies (CISTI)*, Lousada, Portugal, June 2011, pp. 1-6.

[20] `http://httpd.apache.org/docs/1.3/logs.html`

[21] `http://www.rfc-base.org/rfc-1413.html`

[22] D. Jayathilake, "Towards Structured Log Analysis," in *9th International Joint Conference on Computer Science and Software Engineering (JCSSE 2012)*, Bangkok, Thailand, May âĂŞ June 2012, pp. 259 264.

[23] `www.loggly.com`

[24] `www.splunkstorm.com`

[25] `http://wiki.apache.org/hadoop/PoweredBy`

[26] `https://amplab.cs.berkeley.edu/software`

[27] R. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker and I. Stoica, "Shark: SQL and Rich Analytics at Scale," in *SIGMOD 2013*, New York, USA, June 2013, pp. 13-24.

[28] `http://wiki.apache.org/hadoop`

[29] `http://nutch.apache.org`

[30] G. Sanjay, G. Howard and L. Shun-Tak, "The Google File System," in *19th ACM Symposium on Operating Systems Principles*, Lake George, NY, October 2003.

[31] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, San Francisco, CA, December 2004.

[32] `http://hortonworks.com/hadoop/yarn/`

[33] `http://wiki.apache.org/hadoop/PoweredByYarn`

[34] `http://hadoop.apache.org/docs/r2.4.1/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html`

[35] `http://hive.apache.org/`

[36] `http://mahout.apache.org/`

[37] `http://zookeeper.apache.org/`

[38] `http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html`

[39] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters, " *Communications of the ACM*, vol. 51, no. 1, pp. 107-113, 2008.

[40] L. Wang, J. Tao, R. Ranjan, H. Marten, A. Streit, J. Chen and D. Chen, " G-Hadoop: MapReduce across distributed data centers for data-intensive computing, " *Future Generation Computer Systems*, vol. 29, no 3, pp. 739-750, 2013.

[41] `https://databricks.com/spark`

[42] `https://blogs.apache.org/foundation/entry/the_apache_software_foundation_announces50`

[43] `http://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html`

[44] `https://spark.apache.org/`

[45] M. Zaharia, M. Chowdhury, T. Das, Ank. Dave, J. Ma, M. McCauley, M. J. Franklin, Sc. Shenker and Ion Stoica, "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing," in *9th USENIX conference on Networked*

Systems Design and Implementation, CA, USA, 2012, pp. 2-2.

[46] `http://spark.apache.org/docs/latest/programming-guide.html`

[47] M. Zaharia, M. Chowdhury, M. J. Franklin, Sc. Shenker and Ion Stoica, "Spark: Cluster Computing with Working Sets," in *2nd USENIX conference on Hot topics in cloud computing*, CA, USA, 2010, pp.10-12.

[48] `https://okeanos.grnet.gr`

[49] Ev. Koukis and P. Louridas, "okeanos IaaS," in *EGI Community Forum 2012 / EMI Second Technical Conference*, Munich, Germany, March 2012.

[50] `http://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.1.3/bk_using-apache-hadoop/content/yarn_overview.html`